A

MiniProjectReport

On

# ‘LudoGame’

Presented by

*Bhoyar Sakshi Suresh*

SY_CSE[A]

Second Year

Engineering2024-2025

GuidedBy

*MsNitu.L.Pariyal*

(DepartmentofComputerScienceandEngineering)

**Submittedto**

**MGM’sCollegeofEngineering,Nanded**

Under

**Dr.BabasahebAmbedkarTechonogicalUniversity,Lonere**

# Certificate

**Thisistocertifythatthereportentitled**

**LudoGame**

**SubmittedBy**

Bhoyar Sakshi Suresh

insatisfactorymannerasapartialfulfillmentofSY

_CSE[A] inSecond Year Engineering

To

**MGM'sCollegeofEngineering,Nanded**

Under

**Dr.BabasahebAmbedkarTechonogicalUniversity,Lonere**

hasbeencarriedoutundermyguidance,

**Ms.Nitu.L.Pariyal**

**ProjectGuided**

| | |
|---|---|
| **Dr.A.M.Rajurkar** | **Dr.LathkarG.S.D** |
| **HOD** | **irector** |
| DeptarmentofComputerSciencea ndEngineering | MGM'sCollegeofEngineering,Nanded |

## ACKNOWLEDGEMENT

We are greatly indebted to our seminar guided by Ms.Nitu.L.Pariyal for her ableguidance throughout this work. It has been an altogether different experience to workwith her and we would like to thank her for her help, suggestions and numerousdiscussions.
WegladlytakethisopportunitytothankDr.A.M.Rajurkar.(HeadofComputerScience&Engineering,MGM'sCollegeofEngineering,Nanded).

WeareheartilythankfultoDr.LathkarG.S.(Director,MGM'sCollegeofEngineering, Nanded) for providing facility during progress of Mini project; also forherkindlyhelp,guidance andinspiration.

Lastbutnotleastwearealsothankfultoallthosewho helpdirectlyorindirectlytodevelopthisseminarandcompleteitsuccessfully.

WithDeepReverence,

Bhoyar Sakshi Suresh
[SY-CSE-A]

# ABSTRACT

## Tic-Tac-Toe Game

The **Tic-Tac-Toe Game** is a simple console-based application developed in Java, designed to demonstrate basic programming concepts and logic-building skills. This project uses a 3x3 grid where two players alternately place their marks (X or O) to achieve a winning combination of three consecutive marks in a row, column, or diagonal. The game incorporates essential features such as input validation, turn-based gameplay, win condition checks, and detection of a draw.

The primary objective of the project is to explore core Java concepts, including the use of 2D arrays, loops, conditionals, and modular programming by breaking the code into reusable methods. The design ensures simplicity, interactivity, and clarity, making it suitable for both learning purposes and casual entertainment.

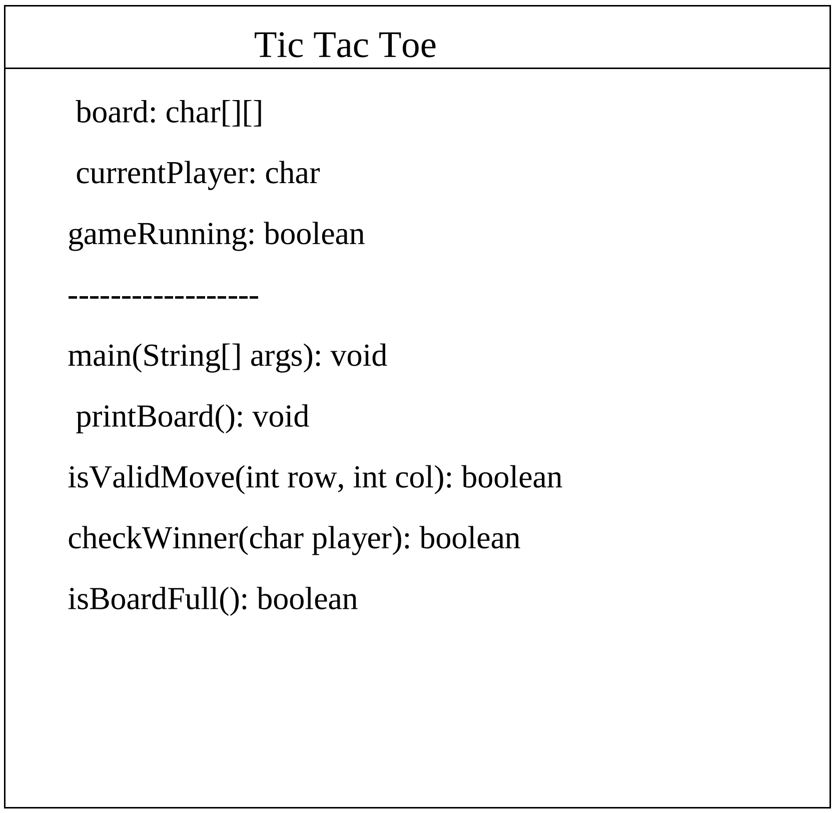
This project can be further extended to include advanced features, such as a graphical user interface (GUI), single-player mode with an AI opponent, and score tracking. Overall, the Tic-Tac-Toe game is an excellent example of how programming fundamentals can be applied to create a functional and engaging application.

Sakshi Bhoyar [142]

SY-A[CSE]

# CONTENT

## 1. UMLDiagram



Tic Tac Toe

```
 board: char[][]
 currentPlayer: char
gameRunning: boolean
------------------
main(String[] args): void
 printBoard(): void
isValidMove(int row, int col): boolean
checkWinner(char player): boolean
isBoardFull(): boolean
```

## 2. Flowchart

Start

↓

Initialize board, currentPlayer = X, gameRunning = true

↓

Display board

↓

Ask for player move (row and column)

↓

```
Decision: Is the move valid?
    If Yes:
        Update the board with the current player's mark
        Decision: Is there a winner?
            If Yes: Announce winner, set gameRunning = false
            Else: Decision: Is the board full?
                If Yes: Announce draw, set gameRunning = false
                If No: Switch player (X ↔ O)
    If No: Prompt for valid move
Repeat steps 5–9 until gameRunning = false
```

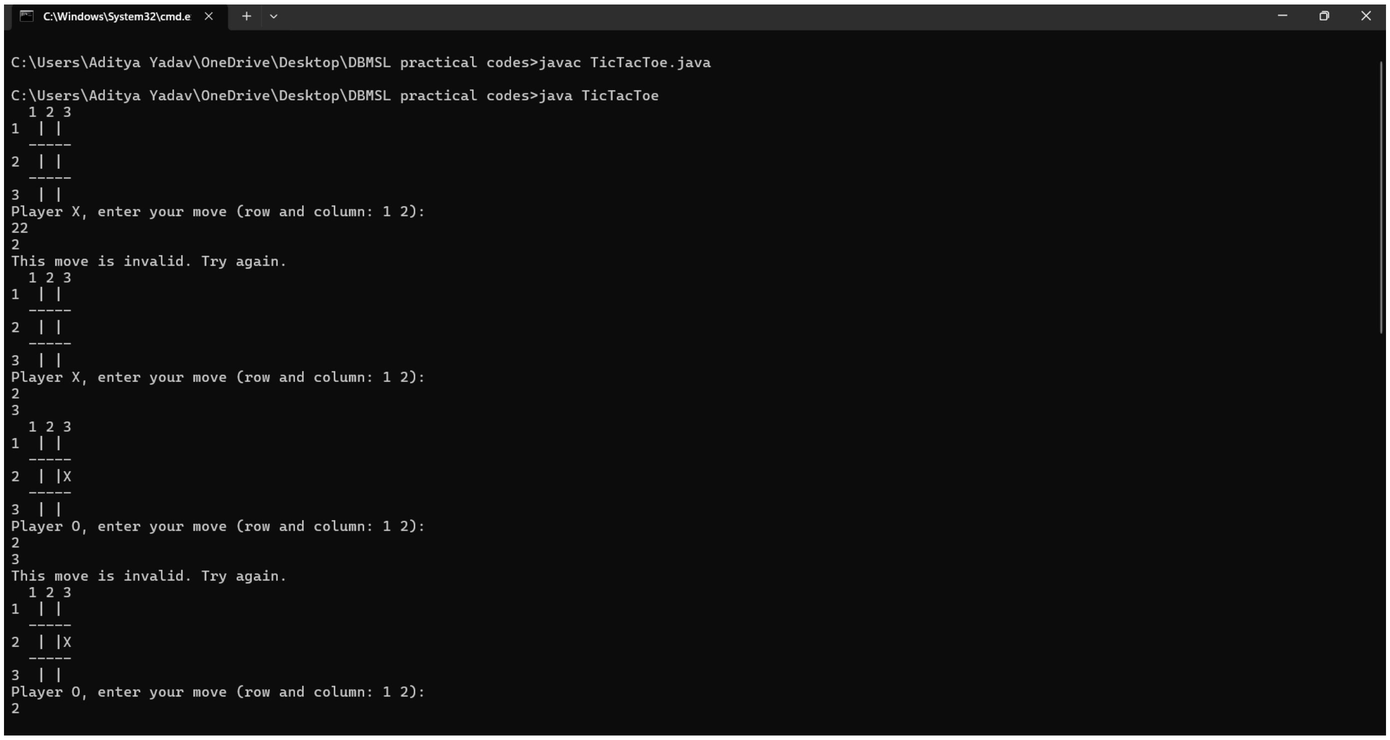
↓

End game and exit

# 3 CodeofTic Tac Toe

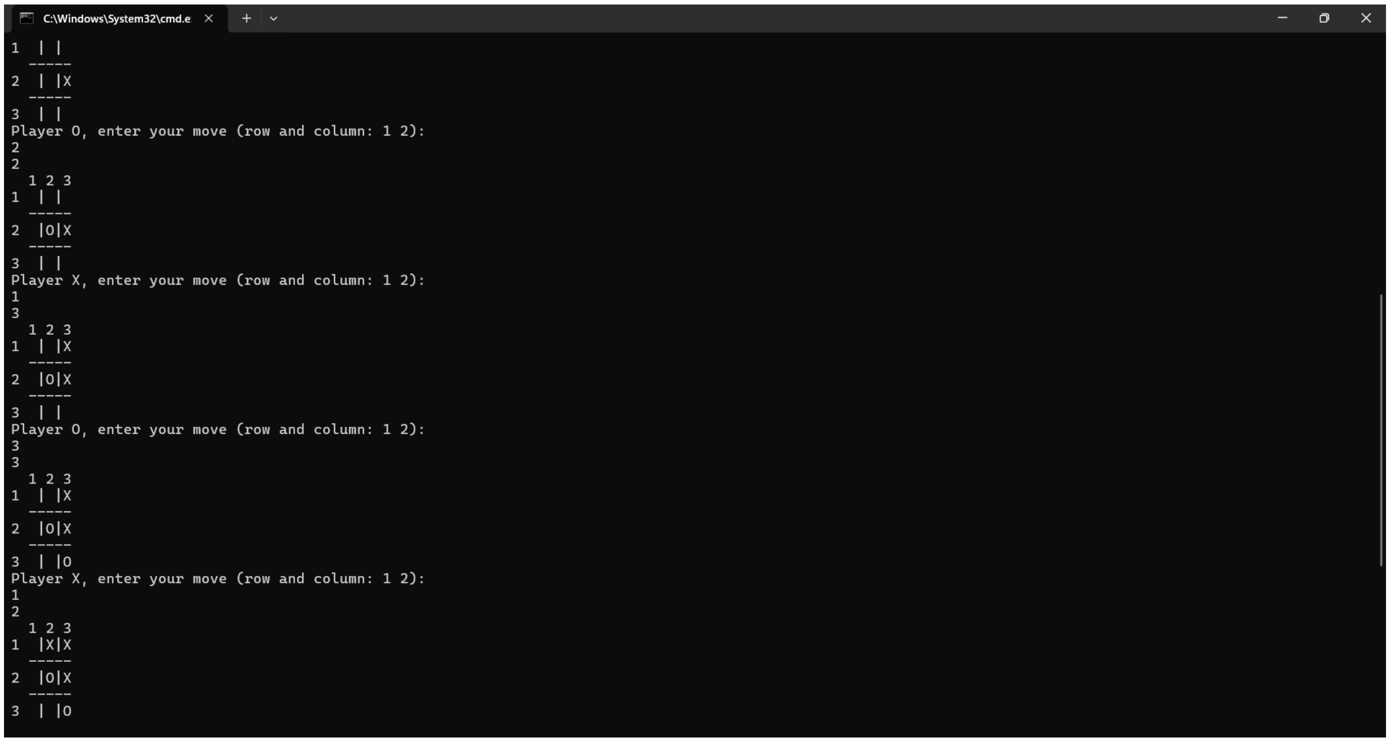
```
import java.util.Scanner;

public class TicTacToe {
    public static void main(String[] args) {
        char[][] board = {
            {' ', ' ', ' '},
            {' ', ' ', ' '},
            {' ', ' ', ' '}
        };
        char currentPlayer = 'X';
        boolean gameRunning = true;

        Scanner scanner = new Scanner(System.in);

        while (gameRunning) {
            printBoard(board);
            System.out.println("Player " + currentPlayer + ", enter your move (row and column: 1 2): ");
            int row = scanner.nextInt() - 1;
            int col = scanner.nextInt() - 1;

            if (isValidMove(board, row, col)) {
                board[row][col] = currentPlayer;

                if (checkWinner(board, currentPlayer)) {
                    printBoard(board);
                    System.out.println("Player " + currentPlayer + " wins!");
                    gameRunning = false;
                } else if (isBoardFull(board)) {
                    printBoard(board);
                    System.out.println("It's a draw!");
                    gameRunning = false;
                } else {
                    currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
                }
            } else {
                System.out.println("This move is invalid. Try again.");
            }
        }

        scanner.close();
    }
    public static void printBoard(char[][] board) {
        System.out.println("  1 2 3");
        for (int i = 0; i < 3; i++) {
            System.out.print((i + 1) + " ");
            for (int j = 0; j < 3; j++) {
                System.out.print(board[i][j]);
                if (j < 2) System.out.print("|");
            }
            System.out.println();
            if (i < 2) System.out.println("  -----");
        }
    }

    // Check if the move is valid
    public static boolean isValidMove(char[][] board, int row, int col) {
        return row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == ' ';
    }

    // Check if a player has won
    public static boolean checkWinner(char[][] board, char player) {
        // Check rows and columns
        for (int i = 0; i < 3; i++) {
            if ((board[i][0] == player && board[i][1] == player && board[i][2] == player) ||
                (board[0][i] == player && board[1][i] == player && board[2][i] == player)) {
                return true;
            }
        }
        // Check diagonals
        if ((board[0][0] == player && board[1][1] == player && board[2][2] == player) ||
            (board[0][2] == player && board[1][1] == player && board[2][0] == player)) {
            return true;
        }
        return false;
    }

    // Check if the board is full
    public static boolean isBoardFull(char[][] board) {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    return false;
                }
            }
        }
        return true;
    }
}
```

**3. OUTPUT**



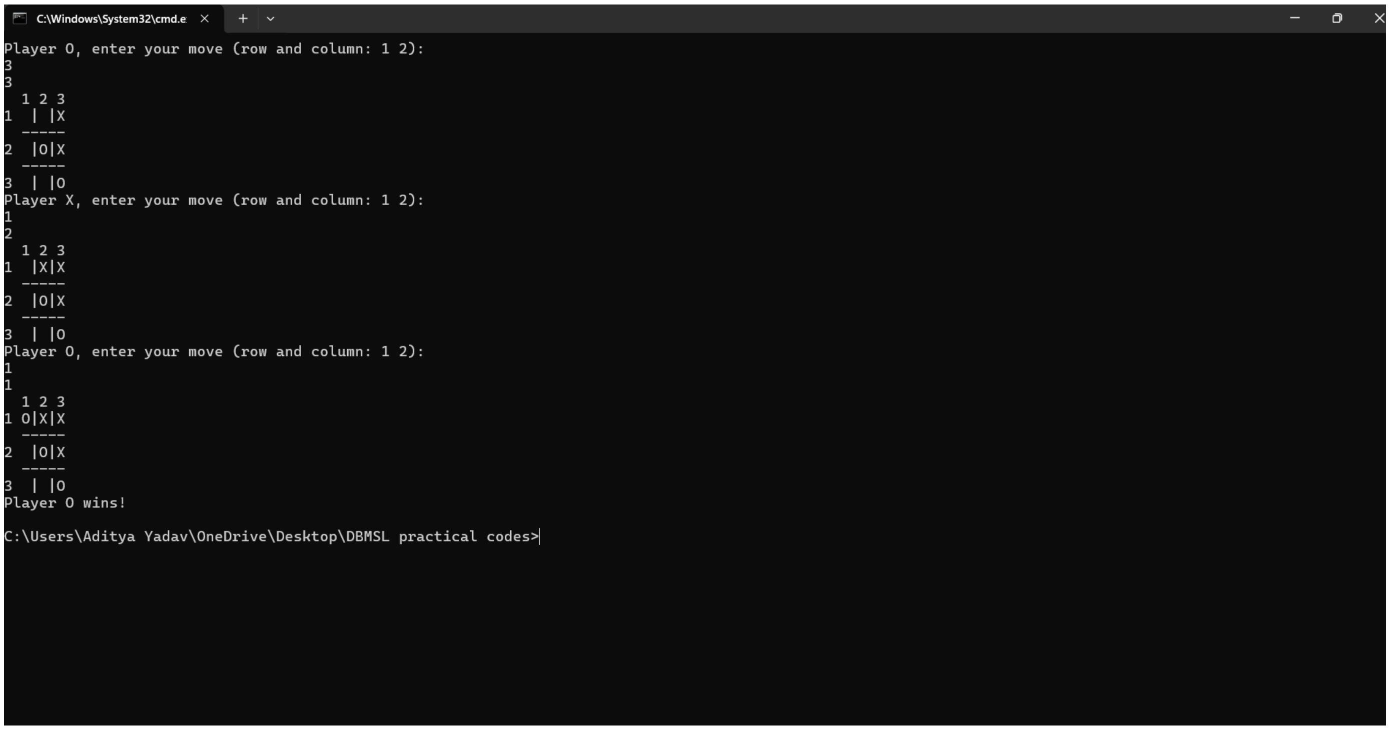
C:\Windows\System32\cmd.e

```
C:\Users\Aditya Yadav\OneDrive\Desktop\DBMSL practical codes>javac TicTacToe.java

C:\Users\Aditya Yadav\OneDrive\Desktop\DBMSL practical codes>java TicTacToe
  1 2 3
1  | |
  -----
2  | |
  -----
3  | |
Player X, enter your move (row and column: 1 2):
22
2
This move is invalid. Try again.
  1 2 3
1  | |
  -----
2  | |
  -----
3  | |
Player X, enter your move (row and column: 1 2):
2
3
  1 2 3
1  | |
  -----
2  | |X
  -----
3  | |
Player O, enter your move (row and column: 1 2):
2
3
This move is invalid. Try again.
  1 2 3
1  | |
  -----
2  | |X
  -----
3  | |
Player O, enter your move (row and column: 1 2):
2
```



C:\Windows\System32\cmd.e

```
1  | |
  -----
2  | |X
  -----
3  | |
Player O, enter your move (row and column: 1 2):
2
2
  1 2 3
1  | |
  -----
2  |O|X
  -----
3  | |
Player X, enter your move (row and column: 1 2):
1
3
  1 2 3
1  | |X
  -----
2  |O|X
  -----
3  | |
Player O, enter your move (row and column: 1 2):
3
3
  1 2 3
1  | |X
  -----
2  |O|X
  -----
3  | |O
Player X, enter your move (row and column: 1 2):
1
2
  1 2 3
1  |X|X
  -----
2  |O|X
  -----
3  | |O
```



```
Player O, enter your move (row and column: 1 2):
3
3
  1 2 3
1  | |X
  -----
2  |O|X
  -----
3  | |O
Player X, enter your move (row and column: 1 2):
1
2
  1 2 3
1  |X|X
  -----
2  |O|X
  -----
3  | |O
Player O, enter your move (row and column: 1 2):
1
1
  1 2 3
1 O|X|X
  -----
2  |O|X
  -----
3  | |O
Player O wins!

C:\Users\Aditya Yadav\OneDrive\Desktop\DBMSL practical codes>
```

## 4.Explanationofcode

**1. Initialization**

java

Copy code

```
char[][] board = {
    {' ', ' ', ' '},
    {' ', ' ', ' '},
    {' ', ' ', ' '}
};
char currentPlayer = 'X';
boolean gameRunning = true;
```

- **board**: A 2D character array represents the 3x3 Tic-Tac-Toe grid. Empty cells are initialized with a space ' '.
- **currentPlayer**: Tracks the current player ('X' or 'O').
- **gameRunning**: A flag to control the main game loop.

---

**2. Game Loop**

java

Copy code

```
while (gameRunning) {
    printBoard(board);
    System.out.println("Player " + currentPlayer + ", enter your move (row and column: 1 2): ");
    int row = scanner.nextInt() - 1;
    int col = scanner.nextInt() - 1;

    if (isValidMove(board, row, col)) {
        board[row][col] = currentPlayer;

        if (checkWinner(board, currentPlayer)) {
            printBoard(board);
            System.out.println("Player " + currentPlayer + " wins!");
            gameRunning = false;
        } else if (isBoardFull(board)) {
            printBoard(board);
            System.out.println("It's a draw!");
            gameRunning = false;
        } else {
            currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
        }
    } else {
        System.out.println("This move is invalid. Try again.");
    }
}
```

- **Main Loop**: The game continues as long as gameRunning is true.
- **Printing the Board**: The printBoard method displays the current state of the board.
- **Player Input**: Players enter the row and column (1-based). The code converts this to 0-based indexing.

- **Move Validation**: The isValidMove method ensures the cell is empty and within bounds.
- **Win Check**: The checkWinner method determines if the current player has won after their move.
- **Draw Check**: If the board is full and no one has won, the game ends in a draw.
- **Player Switching**: After a valid move, the currentPlayer alternates between 'X' and 'O'.

---

**3. Printing the Board**
java
Copy code
```java
public static void printBoard(char[][] board) {
    System.out.println("  1 2 3");
    for (int i = 0; i < 3; i++) {
        System.out.print((i + 1) + " ");
        for (int j = 0; j < 3; j++) {
            System.out.print(board[i][j]);
            if (j < 2) System.out.print("|");
        }
        System.out.println();
        if (i < 2) System.out.println("  -----");
    }
}
```

- **Grid Layout**: Displays the grid with row and column numbers for easy input.
- **Dividers**: Prints horizontal ("-----") and vertical ("|") separators.

---

**4. Validating Moves**
java
Copy code
```java
public static boolean isValidMove(char[][] board, int row, int col) {
    return row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == ' ';
}
```

- Ensures the row and column are within bounds (0 <= row, col < 3) and the chosen cell is empty (board[row][col] == ' ').

---

**5. Checking for a Winner**
java
Copy code
```java
public static boolean checkWinner(char[][] board, char player) {
    for (int i = 0; i < 3; i++) {
        if ((board[i][0] == player && board[i][1] == player && board[i][2] == player) ||
            (board[0][i] == player && board[1][i] == player && board[2][i] == player)) {
            return true;
        }
    }
    if ((board[0][0] == player && board[1][1] == player && board[2][2] == player) ||
        (board[0][2] == player && board[1][1] == player && board[2][0] == player)) {
        return true;
    }
    return false;
}
```

- **Rows and Columns**: Checks each row and column for three identical marks of the current player.

- **Diagonals**: Verifies the two diagonals for a win condition.

---

**6. Checking for a Full Board**

java
Copy code

```
public static boolean isBoardFull(char[][] board) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false;
            }
        }
    }
    return true;
}
```

- Loops through the entire board to see if any cell is still empty (' ').
- If all cells are filled, the game ends in a draw.

---

**Key Features**

1. **Simple Input/Output**: Uses a console-based approach for simplicity.
2. **Error Handling**: Prevents invalid moves or out-of-bound inputs.
3. **Dynamic Player Switching**: Automatically alternates between 'X' and 'O'.

## CONCLUSIONS

The Java-based **Tic-Tac-Toe** game is a straightforward and engaging project that demonstrates fundamental programming concepts, including:

1. **Core Programming Skills**:
    - Handling user input with Scanner.
    - Using loops (while, for) for repeated tasks like board printing and validation.
    - Managing conditional logic with if-else statements for move validation, win detection, and game flow.
2. **2D Arrays**:
    - A great example of applying 2D arrays to represent a grid-based structure.
3. **Game Logic Design**:
    - Implementing rules like checking for winners (rows, columns, diagonals) and detecting draw conditions.
    - Ensuring valid player moves and alternating turns effectively.
4. **Basic Interaction**:
    - A simple console-based interface allows players to interact and see the game evolve step by step.